1. Use the spread (`...`) syntax
2. Use the `Object.assign()` method
3. Use the `JSON.stringify()` and `JSON.parse()` methods

The following illustrates how to copy an object using three methods above:

```
const person = {

   firstName: 'John',

   lastName: 'Doe'

};


// using spread ...
```

```
let p1 = {

   ...person

};

// using  Object.assign() method

let p2 = Object.assign({}, person);

// using JSON

let p3 = JSON.parse(JSON.stringify(person));
```

Both spread (`...`) and `Object.assign()` perform a shallow copy while the JSON methods carry a deep copy.

Shallow copy vs. deep copy

In JavaScript, you use variables to store values that can be primitive or references. When you make a copy of a value stored in a variable, you create a new variable with the same value. For a primitive value, you just simply use a simple assignment:

```
let counter = 1;
```

```
let copiedCounter = counter;
```

And when you change the value of the copied variable, the value of the original remains the same.

```
copiedCounter = 2;
```

```
console.log(counter);
```

Output:

```
1
```

However, if you use the assignment operator for a reference value, it will not copy the value. Instead, both variables will reference the same object in the memory:

```
let person = {

   firstName: 'John',

   lastName: 'Doe'

};

let copiedPerson = person;
```

And when access the object via the new variable (copiedPerson) and change the value of its property (name), you change the value of the property of the object.

```
copiedPerson.firstName = 'Jane';
```

```
console.log(person);
```

Output:

```
{
```

```
   firstName: 'Jane',
```

```
   lastName: 'Doe'
```

```
}
```

A deep copying means that value of the new variable is disconnected from the original variable while a shallow copy means that some values are still connected to the original variable.

Shallow copy example

Consider the following example:

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    address: {
        street: 'North 1st street',
        city: 'San Jose',
        state: 'CA',
        country: 'USA'
    }
};
```

```
let copiedPerson = Object.assign({}, person);
```

```
copiedPerson.firstName = 'Jane'; // disconnected
```

```
copiedPerson.address.street = 'Amphitheatre Parkway'; //
connected
```

```
copiedPerson.address.city = 'Mountain View'; // connected
```

```
console.log(copiedPerson);
```

In this example:

- First, create a new object named `person`.
- Second, clone the `person` object using the `Object.assign()` method.
- Third, change the first name and address information of the `copiedPerson` object.

Here is the output:

```
{
   firstName: 'Jane',
   lastName: 'Doe',
   address: {
        street: 'Amphitheatre Parkway',
        city: 'Mountain View',
        state: 'CA',
        country: 'USA'
   }
}
```

However, when you show the values of the person object, you will find that the address information changed but the first name:

```
console.log(person);
```

Output:

```
{
   firstName: 'John',
   lastName: 'Doe',
   address: {
        street: 'Amphitheatre Parkway',
        city: 'Mountain View',
        state: 'CA',
```

```
        country: 'USA'

    }

}
```

The reason is that the address is reference value while the first name is a primitive value. Both `person` and `copiedPerson` references different objects but these objects reference the same `address` objects.

Deep copy example

The following snippet replaces the `Object.assign()` method by the JSON methods to carry a deep copy the `person` object:

```
let person = {

    firstName: 'John',
```

```
    lastName: 'Doe',

    address: {

        street: 'North 1st street',

        city: 'San Jose',

        state: 'CA',

        country: 'USA'

    }

};

let copiedPerson = JSON.parse(JSON.stringify(person));

copiedPerson.firstName = 'Jane'; // disconnected
```

```
copiedPerson.address.street = 'Amphitheatre Parkway';


copiedPerson.address.city = 'Mountain View';


console.log(person);
```

Output

```
{
   firstName: 'John',
   lastName: 'Doe',
   address: {
       street: 'North 1st street',
       city: 'San Jose',
       state: 'CA',
```

```
        country: 'USA'


    }


}
```

In this example, all values in the `copiedPerson` object are disconnected from the original `person` object.